RD&E

CENTER

# Technical Report

No. 13448

## Telerobotic Control For Teams of Semi-Autonomous Agents

Contract Number: DAAE07-88-C-R076

~~Draft Final Report: March 10, 1989~~
Final Report: ~~April 24, 1989~~

By Marcel Schoppers & Dan Shapiro

Advanced Decision Systems
1500 Plymouth Street
Mountain View, CA 94040
ADS Project Number 3213



U.S. ARMY TANK-AUTOMOTIVE COMMAND
RESEARCH, DEVELOPMENT & ENGINEERING CENTER
Warren, Michigan 48397-5000

89

## NOTICES

This report is not to be construed as an official Department of the Army position.

Mention of any trade names or manufacturers in this report shall not be construed as an official endorsement or approval of such products or companies by the U.S. Government.

Destroy this report when it is no longer needed. Do not return it to the originator.

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a REPORT SECURITY CLASSIFICATION: UNCLASSIFIED | 1b RESTRICTIVE MARKINGS |
| 2a SECURITY CLASSIFICATION AUTHORITY | 3 DISTRIBUTION/AVAILABILITY OF REPORT: Approved for Public Release: Distribution Unlimited |
| 2b DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 4 PERFORMING ORGANIZATION REPORT NUMBER(S): ADS No. TR3213-01 | 5 MONITORING ORGANIZATION REPORT NUMBER(S) |
| 6a NAME OF PERFORMING ORGANIZATION: Advanced Decision Systems; 6b OFFICE SYMBOL (If applicable) | 7a NAME OF MONITORING ORGANIZATION |
| 6c ADDRESS (City, State, and ZIP Code): 1500 Plymouth Street, Mountain View, CA 94043-1230 | 7b ADDRESS (City, State, and ZIP Code) |
| 8a NAME OF FUNDING/SPONSORING ORGANIZATION: U.S. Army Tank-Automotive Command; 8b OFFICE SYMBOL (If applicable) | 9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: DAAE07-88-C-R076 |
| 8c ADDRESS (City, State, and ZIP Code): Attn: AMSTA-RRT, Warren, MI 48397-5000 | 10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO |

11 TITLE (Include Security Classification)

Telerobotic Control For Teams of Semi-Autonomous Agents

12 PERSONAL AUTHOR(S)

Marcel Schoppers, Daniel Shapiro

| 13a TYPE OF REPORT | 13b TIME COVERED | 14 DATE OF REPORT (Year, Month, Day) | 15 PAGE COUNT |
|---|---|---|---|
| FINAL REPORT | FROM 7/19/88 TO 2/27/89 | 1989 April 20 | 34 |

16 SUPPLEMENTARY NOTATION

| 17 COSATI CODES | | | 18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB-GROUP | robotic command center, reactive behavior, survivability, manpower leverage |
| | | | |
| | | | |

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The Army Tank-Automotive Command (TACOM) is developing a Robotic Command Center, which will allow a commander and two drivers (three people) to control four remote vehicles. The drivers will be attempting to control two vehicles each -- a difficult task. Giving vehicles a measure of autonomy is one way to simplify that task. On the other hand, deployed vehicles must not be hampered by software with limited competence or reliability. TACOM needs vehicles that are at once intelligent *and* survivable. Furthermore, the RCC may be operational as early as two years from now. The only solution available in that time frame is a vehicle control interface that allows for the entire range of possibilities between autonomous behavior and continuous teleoperation.

(continued other side)

| 20 DISTRIBUTION/AVAILABILITY OF ABSTRACT | 21 ABSTRACT SECURITY CLASSIFICATION |
|---|---|
| ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS | UNCLASSIFIED |
| 22a NAME OF RESPONSIBLE INDIVIDUAL: LARRY SIEH | 22b TELEPHONE (Include Area Code): 313/574-5440; 22c OFFICE SYMBOL |

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE: UNCLASSIFIED

BLOCK 14 - Cont'd.

We are fusing ideas from telerobotics and from the Reaction Planning subfield of Artificial Intelligence, to develop a technology that permits vehicles to react autonomously to expected circumstances, while also permitting the human operator to instruct vehicles about what to do and how to do it. In this Phase I study we have programmed some examples of behaviors for the vehicles to perform semi-autonomously, and have tested those behaviors by running them in the TeamWorks I environment. We have also designed and built an interface that allows the vehicle driver to communicate with the intelligent software controlling any and all of the vehicles. In this way we have been able to demonstrate that a single person is capable of controlling four simulated vehicles at the same time. In Phase II we intend to expand the competence of the vehicles, to give the operator new ways of instructing the vehicles, and to push those capabilities toward deployment in the real world.

# TABLE OF CONTENTS

# List of Figures

## 1. Introduction

This Final Report, prepared by Advanced Decision Systems (ADS) for the U.S. Army Tank Command (TACOM) under Phase I SBIR Contract DAAE07-88-C-R076, describes progress made toward a robotic vehicle control scheme that allows for the entire range of possibilities between autonomous behavior and teleoperation.

## 2. Objectives

The Army Tank Command has long term goals of increasing fighting power and protecting manpower. Those goals come together naturally in the Robotic Command Center (RCC), which will utilize advanced technology in such a way that a small number of personnel can control a larger number of vehicles from a distance. In its present realization, a commander and two drivers will be equipped to control four Wiesel vehicles as they engage in typical surveillance and combat maneuvers (greater manpower leverage factors are also of interest). At issue is the degree of machine independence. The RCC project has the charter to explore a spectrum of control strategies, ranging from teleoperation of vehicles, through telerobotic command for guiding predefined vehicle tactics, to complete vehicle autonomy governed indirectly through a set of goals. The desired result is a demonstration of robot vehicle technology with clear practical application in 5-10 years.

This time frame restricts the possible solutions. In particular, given the current state of the art in artificial perception and intelligence systems, three approaches are possible:

1. build an autonomous system with very limited capabilities[1] ,
2. teleoperate the vehicle, or
3. design an interactive system which allows for the entire range of possibilities between autonomous behavior and teleoperation.

Our work addresses the third option. Utilizing a simulation as the execution environment, we are developing an interactive system that supports man-in-the-loop sensing and acting decisions, while providing a command language for robot instruction that can grow through time. We are demonstrating control of multiple vehicles by a single user (showing manpower leverage), reactive execution, arbitration among many behaviors, and use of the behavior vocabulary as a command language. In short, we are

---

[1] The best current examples of autonomous vehicles possess restricted road following algorithms, very little cross country capability, and almost no model of the environment beyond "go" and "no go" terrain. Their perceptual repertoire does not include (and cannot easily be made to include) recognition of buildings, other vehicles, rivers, landmarks, people, trees, fences, threats, or cover, etc. Furthermore (and probably because of such perceptual limitations), these vehicles have no vocabulary of intentions beyond simple maneuver goals.

working towards a vehicle control interface usable in the RCC. Phase II goals are discussed in more detail in Section 5.7.

## 3. Conclusions

Our Phase I effort was aimed at clarifying the issues and solutions required to achieve the Phase II objectives. We did this with a demonstration of reactive behavior. Building on the Teamworks simulator developed at ADS for the Army Tank Command [7], we demonstrated that:

- vehicles can be instructed, and left to follow those instructions as best they can, thus allowing the controlling personnel to manage several vehicles in parallel;
- a team of vehicles can be managed either individually or as a team, so that the commander can, with a single instruction, manage the entire team;
- vehicles can be programmed to recognize their own limitations, and to ask for help when they are (for example) having difficulty executing their instructions, or when they are incapable or insufficiently informed to make an approaching decision;
- various functions of the vehicles' intelligence can be independently turned on and off, thus allowing a human operator to teleoperate parts of the vehicles without also interrupting other autonomous functions.

This report describes a software demonstration of a control interface. The person selected formations, pointed out destinations and waypoints, and determined the degree of autonomy. The vehicles used onboard sensors and reactive artificial intelligence to navigate themselves through previously unknown terrain. This capability was achieved two new ideas known as Universal Plans and Behaviors. To demonstrate the vehicles' reactive behavior the simulation allowed human observers to modify the terrain at any time and at any location by means of simulated bomb craters, which became surprise obstacles for the vehicles. The human observer could also cause enemy aircraft to appear overhead at any time. The demonstration showed that despite the vehicles' having to move through unfamiliar and dynamically changing terrain, the vehicles' reactive intelligence made it possible for one person to direct the navigation of four vehicles moving in formation at simulated speeds of up to 20 miles per hour. The demonstration utilized the simulated environment of the TeamWorks (Phase I) testbed for tank platoon strategizing.

## 4. Recommendations

Phase I has demonstrated that there is significant manpower leverage to be had from semi-autonomous reactive vehicles, but did so in a context of a small library of behaviors, an incomplete language for expressing behaviors, a limited range of ways of controlling and combining behaviors, and an unrealistic simulation. To move our concept to practical utility, all these factors will have to be reversed. That is, follow-on work should:

- extend the behavior representation and execution languages;
- enlarge the competence of the robotic vehicles;
- engineer away certain technical problems like directional oscillation;
- allow vehicles to communicate with and observe each other;
- discover principles that will streamline the control of concurrent activities and detect conflicting instructions;
- develop the vehicle control interface for greater flexibility in robotic instruction and reprogramming;
- test the work in a simulation of more realistic vehicle physics;
- extend the simulation to allow varieties of surprise obstacles, threats, and vehicle formations.

## 5. Discussion

### 5.1. Guide to Reading

The remainder of this report is divided into sections that discuss problems of the application domain, Section 5.2; our vision of how a team of semi-autonomous agents might be controlled, Section 5.3; the approach we are taking, Section 5.4; the technolgy we are employing in both Phase I and Phase II, Section 5.5; the demonstration we produced and presented to the Army Tank Command, Section 5.6.2; and suggestions for future work, Section 5.7. An Appendix contains Minutes of the meeting where ADS presented the final project demonstration.

### 5.2. The Technical Problem

The task of developing a robot command center (subject to the above objectives) requires the following solution kernels:

- a software architecture that supports the desired range in vehicle autonomy, instruction, and teleoperation,

- a user interface that provides effective control of multiple vehicles,
- a method of acquiring the necessary sensor data, and
- the ability to express and execute vehicle plans which can control semi-autonomous vehicles.

This last item is key, because it stresses current robotics technology. To amplify, the domain of controlling semi-autonomous vehicles has the following features:

- the details of the environment are not completely understood in advance (for example, the location of obstacles to movement),
- the situation can change because of the actions of independent agents (enemy weapons), and
- one's own actions are not guaranteed to have the desired effect (for example, the vehicle may slip when told to turn).

These statements are unremarkable in that they can be made about many domains, but work in robotics within Artificial Intelligence has almost exclusively relied on planners that require absolute knowledge of their environment and absolutely reliable actions. As a result, the *a priori* prescriptions for action developed by classical planning systems (in the model of Strips [1]) cannot be utilitized in this domain. These domain characteristics essentially define reactive execution problems, which are the subject of an emerging subfield of AI.

The difference between classical and reactive plans can be understood with a simple example: the task of driving to work. In a non-reactive view, your path might be planned as a dense sequence of coordinate points on a map. This approach requires an extremely accurate model of the world, enough to allow you to execute your path *with your eyes closed* while maintaining safety. *Everything* must be planned out in advance, including when to stop, and how long. You drive by assuming that your actions are having their desired effect.

Reactive plans explicitly encode the need for sensory information, as well as responses to the situations that might arise. So, for example, you know to react to red traffic lights by stopping, to police sirens by pulling over, and to road construction by carefully executing detours. In uncontrolled environments, reactive plans are required.

All of the technical approaches for building reactive execution systems (programs that interpret and execute reactive plans) employ a tight sense-think-act loop as the agent's activity cycle. They vary, however, in the extent to which they expect the world state to conform to the plan. The least reactive approaches are analogous to flow charts; a specific test is applied at a specific time. The most reactive methods execute a continuous loop where the state of the environment is diagnosed and the appropriate

plan step retrieved (i.e., there is no expectation concerning the state of the environment). This latter extreme requires a great deal of sensory information.

Our work attempts a middle ground between these extremes by allowing complete reactivity within specific activities, but with careful control over the activities being performed at any given time. So, for example, the robot might be in "drive to work" mode, with the attendant reactions and sensory needs, but explictly not concerned with the details of "navigate detours". That "mental context" becomes active on the presence of particular cues.

The Robot Command Center has the additional advantage of the presence of a human in the loop. This opens up the possibility of sharing the planning effort between man and machine, and makes the TACOM problem much more tractable. (We note in Section 5.7 that the difficult sensor interpretation tasks can be shared as well).

## 5.3. A Scenario

As an overview, we think it appropriate for TACOM to envisage capabilities of the following kind. A single commander is presented with the problem of controlling an entire tank platoon. The platoon's task is to clear a village of a potential enemy threat. In a testbed environment, the goal is to develop strategies for directing the tanks, while in a live context, the object is to execute an existing strategy with a minimum of required intervention on the part of the commander. Assuming a testbed environment, we imagine the commander first specifying a plan in terms of behaviors known to the vehicles. He will do this by means of a computer console with a specialized control interface. That interface will provide him with a list of relevant commands, and will also display the vehicles' answers and requests. The interaction might go as follows:

1. Commander issues a *Move* command, then points to a map, picks out the South entrance of the village.
2. Commander issues a *Then*, followed by *NewCommand*, and types "Sniping" (to indicate that once the vehicles reach the village they should conduct sniping activities).
3. Commander issues an *Until/Time*, types in 1800 hrs.
4. Commander issues a *Move* followed by another point on the map (meaning that at 6pm the vehicles should leave the village for the new point).

The system might respond that *sniping* is an unknown keyword, and request a definition. The commander could reply by selecting the following sequence of commands:

*DefineAs Scurrying Shooting InParallel*

Here, the *scurrying* behavior is a predefined endless loop composed of *movement to cover* followed by *look for cover*, and the *shooting* behavior is defined with two conditionals: *scan for target* if no target is known, and *shoot at target* if a live target is seen. For the sake of argument, we will assume that the vehicle must stop in order to shoot at a target.

On receiving this definition, the system would automatically identify a conflict based on its analysis of the predefined behaviors:

> There is a conflict between *scurrying* and *shooting*. When a target is available and I (the tank) am moving, would you like me to stop and fire, or move to cover?

While the details of this illustration can change, the key points are that

- The commander communicates in very abstract terms such as "move to the South entrance of the village". The commander does not have to tell the vehicle what detailed path to follow, how fast to go, or what collisions to avoid along the way. Those details are negotiated automatically by virtue of the reactivity in the vehicle's behaviors.
- The commander composes plans by combining behavioral pieces. The vehicle is not limited to the set of capabilities with which it was deployed, and is correspondingly less vulnerable to unforseen circumstances. Further, the vehicle itself aids in the instruction process, e.g. by informing the commander of previous instructions he may have forgotten, or as in the above example, by asking the commander to decide how the inconsistent instructions should be interpreted. These capabilities are provided by the way behaviors are represented and organized.

The piece of the above scenario that is within reach of being demonstrated on the RCC (as designed by FMC) at the end of our Phase II work is interactive navigation with constraints. With a rudimentary navigation ability in place it will become possible to inform the robotic vehicles about various constraints such as the timing of their moves and the environmental conditions under which various activities should take place.

Rather than a driver having to be fully engaged in controlling a vehicle, the driver will instruct the vehicle about the point to be reached, and about points to be avoided along the way, after which the vehicle drives itself either until it recognizes it needs help, at which point it requests help, or until the driver decides to apply some preventive medicine. Meanwhile, however, the driver has been free to communicate with other vehicles. Depending on how competent the vehicles can be made about driving themselves, it may be possible to have one driver manage all four vehicles. Ultimately, in the far-off future, it might be possible to dispense with drivers altogether, leaving only the commander to delegate and coordinate.

When vehicles are informed about constraints on the timing of their moves, for example, the vehicles will be able to take the initiative about what to do and when to do it, and the drivers will enforce navigational quality control.

## 5.4. Technical Approach

Our technical approach to the construction of the RCC prototype relies on the following key ideas:

- simplification of vision processing by use of user assisted scene interpretation [4],
- providing reactivity by encoding vehicle actions in terms of *Universal Plans*, which are highly reactive procedures that achieve agent goals *over a wide range of input situations*,
- achieving execution efficiency by packaging these Universal Plans into contexts, called *Behaviors*, which specify action options, persistent sensing and resource requirements, relevant events, active user commands, and termination criteria,
- providing instructability by using the vocabulary of vehicle behaviors as user commands, and
- integrating these capabilities via a software architecture that allows plan steps and reactions to be interleaved, and which provides surrounding arbitration, and scheduling functions.

This architecture is shown in Figure 1. Its main feature is the *current program* abstraction which is a constantly modified data structure that contains all the behaviors (with associated Universal Plans, sensor tasks, event triggers, etc.) that are active on the agent's mind. As circumstances, or steps in the plan are accomplished, the contents of the current program will change.

The use of a single abstraction containing the agent's state of mind promises several advantages for the future. In particular, it supports exhaustive run-time approaches to arbitration between multiple behaviors, and it suggests use of knowledge intensive techniques for reasoning about actions, given this representation of the agent's current intentions.

## 5.5. The Technology

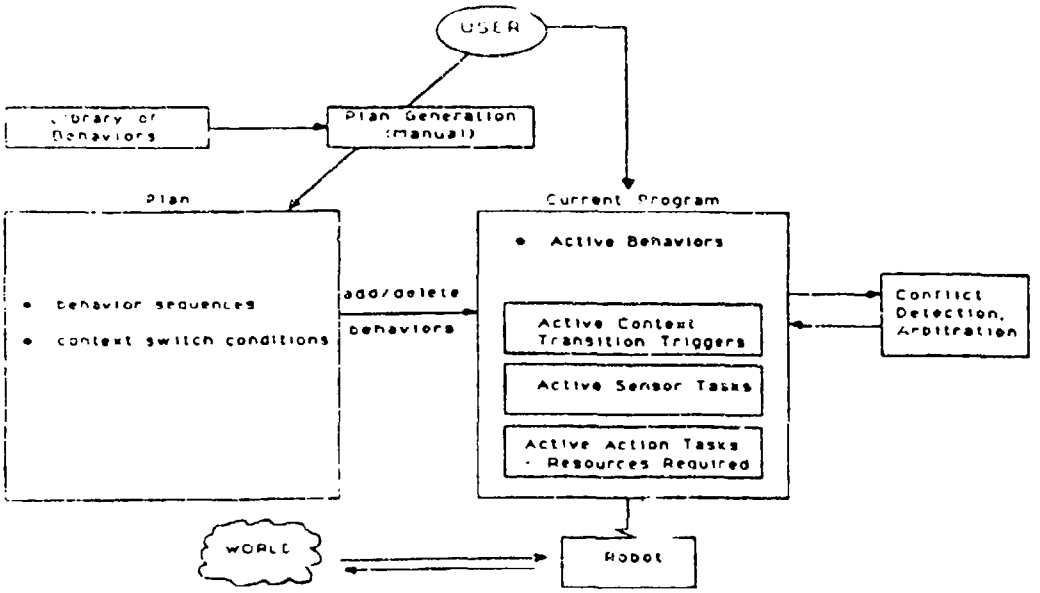


**Figure 1:** A Reactive Execution Architecture

### 5.5.1. Universal Plans

Until recently, AI work on planning was strongly in favor of the predictive planning approach; hence the prominence of planners devoted to solving problems in a world of toy blocks, where nothing moved unless the planner willed it. But in real life, even such toy worlds might be inhabited by mischievous babies that occasionally overturned the planner's block towers. For every such interference, traditional planners have to resort to replanning, predicting a new future. Many of the old, incorrect predictions are no longer relevant, and the planning that went into making those predictions was a waste of time. By the same token the new predictions may also be wrong, and the replanning itself largely a waste of time.

Clearly, the predictive approach becomes more and more inefficient as the environment becomes more dynamic. Indeed, the Blocks-and-Baby problem just mentioned can be made impossible for any predictive planner: suppose there is nothing the baby cannot do to thwart the planner at every instant. Although extreme, this is actually closer to reality than the predictability of the domains usually dealt with in AI. When driving a car to work one does not decide in advance how fast to go, and in what lane, at every moment. Among other things, one never knows what lanes might be occupied by Sunday drivers, nor when traffic lights will be red along the way. One simply responds to whatever one finds at each moment.

In the last three years, the principal investigator for this work has devised a radically new approach to automatic planning, an approach identified as "Universal Plans" [5].

Universal Plans revise the very concept of "plan", on the grounds that any sequential plan is inherently committed to a very specific future. Reliable prediction being notoriously difficult, Universal Plans rely instead on reaction: what a Universal Plan does depends entirely on which situations actually arise. Hence Universal Plans are able to cope not only with failures of actions to achieve intended effects, but also with serendipity and sabotage, as when one robot unintentionally helps or hinders another robot's progress. In particular, this flexibility is obtained without any need for replanning.

A comparison between Universal Plans and process/servo control algorithms is quite tenable. No matter what other traffic does along the way to work, for example, a Universal Plan will "push" the world toward the goal of arriving at the office by doing the appropriate thing at each moment, like overtaking another car or stopping at a red light. A Universal Plan specifies not the order in which things are to be done, but what to do if a certain situation should arise. Whether the same situation (a red light) arises once or twice or thrice, the same response must occur. By contrast, traditional planners must know in advance exactly how many times that situation will arise. Thus, Universal Plans encode adaptive behavior, not specific courses of action with beginnings and ends.

Universal Plans emerged from the principal investigator's earlier experience with programming a mobile robot for autonomous navigation (described in [3]): the code used to drive that robot turned out to exhibit the structure now codified as Universal Plans. But reactivity, while crucial for robotic applications, is not sufficient by itself: if Universal Plans had to be programmed by hand they would amount to little more than process controllers or highly conditional servo routines, which also react according to the situation at each moment. Our goal was to build such programs automatically, i.e. with AI planning software. That that is indeed possible has been established by the principal investigator's PhD work.

Consequently, Universal Plans are truly "plans" in the sense that they can be automatically constructed, their robotic suitability and reactivity notwithstanding. Our work might therefore be viewed as automating the synthesis of process controllers. More generally, we have succeeded in eliminating the boundary between programs that can be reactively executed and programs that can be automatically constructed, and in that respect Universal Plans are unique.

The input to Universal Plans execution is sensory data such as forces and torques on a vehicle as a whole, the position of and torques on a robot arm, and contact signals from sensors in a vehicle's bumper or in a robot hand. On the basis of such information, Universal Plans can determine whether

- the vehicle is moving as desired, and if not, how to correct it;
- the robot arm is carrying a light or heavy object, and is in the right place
- the vehicle has bumped into something, or its hand has grasped the object to be picked up.

That is, sensory information of this kind can be used to create a feedback loop for servo control (as will be shown below).

When a vehicle is being remotely controlled, such reactive feedback servoing will allow the remote operator to communicate in more convenient terms than would otherwise be possible. The operator may be able to initiate the procedure for "make a 3-point turn" -- and then think about something else -- where otherwise s/he would have had to supervise every part of the turn. Similarly, instead of having to explicitly control all movements of the vehicle and its robot arm during a mine-detection search, the operator may be able to tell the vehicle to "probe for mines", causing the vehicle to deploy its robot arm and move both the arm and the vehicle so as to achieve a continuous search pattern. That is, Universal Plans facilitate the move from teleoperation to telerobotics by reacting to sensory information, thus decreasing the communication bandwidth required between the operator and the robot, and freeing the operator to manage other vehicles simultaneously.

Universal Plans alone would be very useful to TACOM for their continuous reactive behavior, and for the burden they remove from operators. A capability for automatically constructing Universal Plans makes them further suited to TACOM's needs. Our intent is to put what we know about automatic construction at the disposal of the operator, both for pre-mission programming and during missions. An example of the use of automatic synthesis techniques is the sniping behavior, described in Section 2 ("Problem Identification"). The sniping behavior involves a composition of two previously independent behaviors now to be executed in parallel. Other ways of combining behaviors are readily imaginable. The point here is that Universal Plans come with program synthesis support facilities largely worked out. This permits the operator to invoke, combine, and arbitrate between desired behaviors, without always having to reprogram the robot first.

Universal Plans are equally important for facilitating interactive supervision of an intelligent robot's reasoning processes. If human beings are to monitor a robot's behaviors and reasoning processes it follows that the robot should be succinct. Previous AI planners have been anything but succinct, planning out the entire course of events arbitrarily far into the future, even when there was no guarantee that the planner's expectations would ever be realized. This made planning and replanning exceedingly tedious for human observers. Universal Plans decide what should be done *now*, and can make that decision without having to reason about a detailed future. "Planning only as much as is necessary to choose the next action" is precisely the kind of succinctness that human observers/operators can tolerate.

We now provide an example of Universal Plan interpretation. The plan described here is well below the level of abstraction adopted by previous AI work, but such low levels of abstraction are essential to interfacing with real robots. The example is: moving a robot arm horizontally from position $x_{now}$ to position $x_f$ in the shortest possib time (we assume Cartesian coordinates).

Suppose the robot arm is equipped with sensors that return integer readings for the horizontal position, speed and acceleration of the hand. The values (e.g. speed and acceleration) sent to the robot motors are also integers. In between, the most primitive actions are numeric functions. For example, we define a **speed-up** action as in Figure 2a. The functions for a **slow-down** stage are similar.

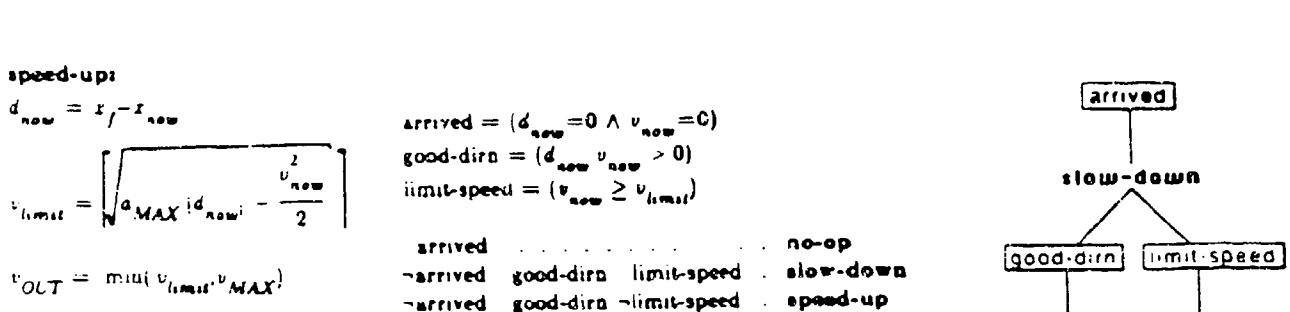


**Figure 2:** Implementing Arm Motions with Feedback.

Whenever we get new values for $x_{now}$ or $v_{now}$, the arm's speed and acceleration can be adjusted to suit. Of course, $x_{now}$ and $v_{now}$ are constantly changing. Thus, computing new arm motion parameters from current sensor readings establishes a feedback loop.

We must also specify when each stage should be used (Figure 2b). There are three relevant predicates: is the arm at its destination? moving in the right direction? moving as fast as its destination will allow? Depending on the truth or falsity of these predicates, one of **slow-down** and **speed-up** will be imposed between sensors and effectors.

Notice that these primitive actions are very different from traditional plan primitives in that they represent I/O conditions to be maintained for some unspecified length of time, not conditions to be achieved in the world. How long a given constraint is in force depends entirely on the environment.

Moreover, there is no particular order in which **speed-up** and **slow-down** must be executed: that too is determined by the environment. Indeed, if the arm was intended to

stop above some specific object and that object is being moved even as the arm is approaching ($x_f$ is changing), the arm will nevertheless do the right thing: by accelerating and decelerating as necessary, the arm will track the moving object until the object comes to rest.

By design, Figure 2b can be expressed as the "Universal Plan" of Figure 2c. This is only a trivial example of a Universal Plan, however. Its purpose is to show that even the most primitive sensori-motor feedback constraints can be controlled by Universal Plans. The plan representation contains additional constructs not detailed here, and is in fact rich enough to support sequential behavior of any desired complexity.

In summary, Universal Plans possess a number of features that make them well suited to telerobotic control. First, Universal Plans are a way of expressing the dependence of robotic response on the behavior of the robot's surroundings. By acknowledging and using that dependence, Universal Plans both release remote operators from the burden of continuous monitoring and allow operator intervention. This potentially frees up the operator to issue instructions to several vehicles at once. Second, Universal Plans are amenable to automatic verification and synthesis techniques, providing a strong base on which to erect interactive specification of behaviors as required for convenient telerobotics. Third, Universal Plans have explicit goal structures and a succinct planner, making it much easier for an operator to inspect a robot's reasoning processes.

### 5.5.2. Behavioral Planning

The following material presents an approach to the construction of reactive plans based on the metaphor of planning as program generation. In particular, we hope to provide users with a significant, online capability for modifying an agent's program/plan using a vocabulary of control primitives and behavioral routines (*behaviors* are procedures of the kind described in the scenario in Section 5.3). That is, we wish to support *instructability*. (The benefits of this approach to man-machine interaction should be obvious.)

In order to support this style of interaction, we rely on machine understandable representations for the structure of plans. Given this knowledge, we believe we can design a limited program synthesis capability which is able to combine behaviors/plans (in interesting ways) as per a user's request.

Figure 3 outlines an architecture for our approach to behavioral planning. This Figure distinguishes several major components: a set of primitive behaviors and a library of plans built from them, an expression of the program for action, a behavior monitor, an execution controller and a mechanism for performing all modifications to the program/plan.

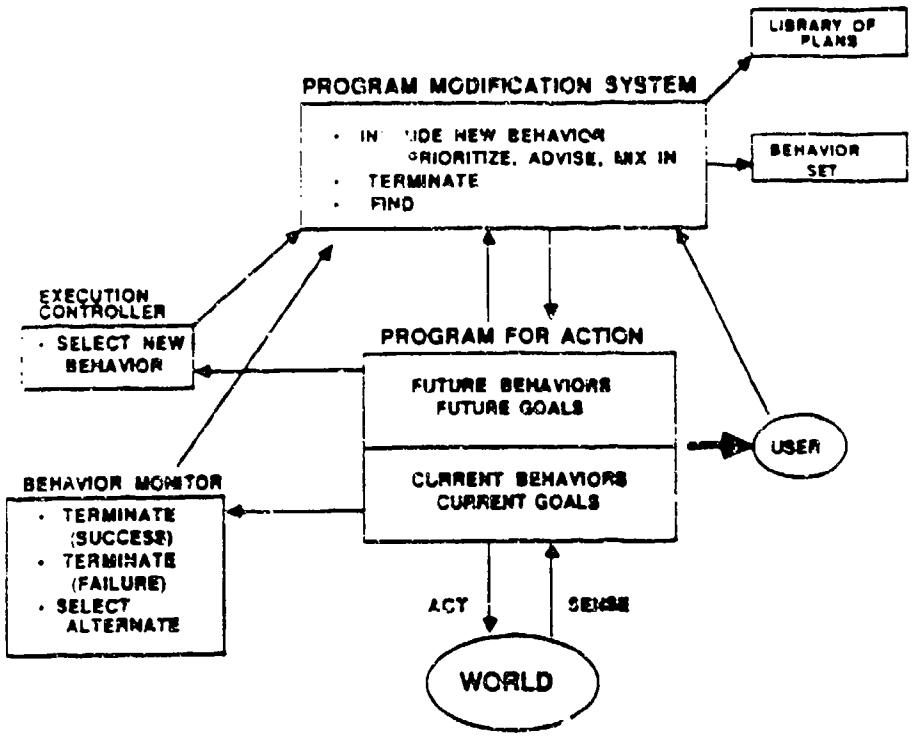


**Figure 3:** An Architecture for Behavioral Planning

The program for action is intended to represent both the current plan (meaning the set of active behaviors) as well as the decisions about how to act in the future to the extent that they are known. This information may be present in the form of goals, or anticipated behaviors (for example, we can preplan our commitment to a movement action before we have terminated a sniping behavior). In the case of purely reactive agents, the future plan may be completely absent.

The segregation of current and future plans provides a basis for separating other control functions. The *execution controller* is responsible for merging applicable portions of the future plan into the current program for action (selecting behaviors to accomplish goals as required), while the *behavior monitor* is responsible for determining if the current plan is executing as desired. The monitor has several corrective actions: it can terminate a running behavior that has successfully accomplished the goal to which it was applied, it can terminate with failure, it can select a new behavior to substitute for a failed one (e.g., replacing a *follow the road* behavior with a *cross country movement* behavior when the road edges become sufficiently obscure), or it can identify an additional behavior to respond to new environmental conditions (if communication is failing, *seek high ground*).

The *program modification component* is the data abstraction for manipulating the plan, and its options reflect the desired vocabulary for combining program elements.

These are also the primitives that support instructability. The complexity of this vocabulary is limited both by the human interface need to keep instructions simple, and by the technology of automatic programming which will be required as support. To set context, the expected scenarios revolve around taking advice in the context of larger programs. For example, the user might instruct the system to "use a column formation" while executing the plan for exploring a forested area. This concept of man machine interaction fits with our understanding of the application domain. Given the time sensitivity of combat operations, it would make sense for plans to be precomputed (and analyzed) to the extent possible. This suggests that advice taking and plan modification will be common processes, while plan generation from goals and priors will be comparatively rare. We expect to create a library of plans/programs which define the tasks each agent commonly performs.

### 5.5.3. Combining Behaviors via Universal Plans

The task of combining behaviors can be viewed as a limited exercise in automatic programming. The difficulty of this task depends in turn upon the control structures allowed in the language for expressing programs/plans. Existing reactive planning systems employ sequence, parallelism, conditionals, subsumption, and pattern directed invocation, as well as rigid prioritization to resolve conflicts among parallel actions. Our choice for control primitives is driven by the desire to provide instructability, and appears to match well with the structures supplied by Universal Plans: sequence, conditionals, pattern directed invocation, and conflict resolution by sequencing activities.

With Universal Plans as the language for expressing behaviors, certain combination mechanisms are simple to define. Indeed, we already have some idea of how parallel execution and prioritizing can be achieved. The following example sketches the Universal Plans realization of the *sniping* behavior.

*Scurrying* means moving from one area of cover to another, and requires knowing where the next area of cover is. If that is not known, the vehicle must look for some cover. Figure 4a shows a decision table that is equivalent to the universal plan for *scurrying*. *Sniping* means shooting at a target, and requires knowing where the target is, and (for the sake of argument) being stationary. Figure 4b shows a decision table for *sniping*. Running the two plans in parallel effectively generates a new decision table (Figure 4c), which shows that under some conditions, *move-to-cover* and *stop* will be executed at the same time. Detecting conflicts of this sort is trivial, and is already part of the Universal Plans synthesis machinery. Once the conflict is discovered, it remains only to decide which of *scurrying* and *sniping* should receive priority. We choose to stop and shoot. This may be encoded with a rule that records both the circumstances in which the conflict arises, and what to do about it:

`next-cover-known` $\wedge$ `target-known` $\rightarrow$ *shooting*

| covered | know-next-cover | move-to-cover | look-for-cover |
|---|---|---|---|
| 1 | 1 | 1 | |
| 1 | 0 | | 1 |
| 0 | 1 | 1 | |
| 0 | 0 | | 1 |

a

| stopped | know-target | sheet | look-for-target | stop |
|---|---|---|---|---|
| 1 | 1 | 1 | | |
| 1 | 0 | | 1 | |
| 0 | 1 | | | 1 |
| 0 | 0 | | 1 | |

b

c

| covered | know-next-cover | stopped | know-target | move-to-cover | look-for-cover | sheet | look-for-target | stop |
|---|---|---|---|---|---|---|---|---|
| - | 1 | 1 | 1 | 1 | | 1 | | |
| - | 1 | 1 | 0 | 1 | | | 1 | |
| | 1 | 0 | 1 | 1 | | | | 1 |
| - | 1 | 0 | 0 | 1 | | | 1 | |
| - | 0 | 1 | 1 | | 1 | 1 | | |
| - | 0 | 1 | 0 | | 1 | | 1 | |
| - | 0 | 0 | 1 | | 1 | | | 1 |
| - | 0 | 0 | 0 | | 1 | | 1 | |

Figure 4: An Example of Behavior Combination

Now it remains only to modify the interpreter so that this rule will be found when the conflict situation arises.

While Universal Plans permit a parallel representation of plans, they do *not* support

- parallel execution;
- prioritizing one parallel activity over another;
- execution-time modification of numerical parameters;
- cliches (see below).

We expect to extend Universal Plans in these regards.

### 5.5.4. Cliches and Behavior Combination

Cliches capture frequently used knowledge, and function as likely solutions to common problems. It may be possible to discover cliches in the robot vehicle domain. In the programming realm they have been employed by the Programmer's Apprentice Project at MIT, with which the authors are intimately familiar [6]. The Programmer's Apprentice has cliches for useful modules and strategies such as list insertion modules, sort routines, queue and process strategies, divide-and-conquer strategies, etc. Discovery of such cliches permits faster solution of novel problems, and also increases the scale of solvable problems.

This abstraction process can be continued to another level. We can potentially define *domain cliches*, not recognized in the Programmer's Apprentice, which describe categories of vehicle behaviors. Some reasonable examples are as follows:

- Define a *physical action*, as a behavior that requires a specific resource, has necessary preconditions, and expected postconditions.
- Define a *careful action*, as a behavior for sensing the precondition of an arbitary action, invoking (in a pattern directed sense) a behavior for establishing the precondition of that action if it isn't already true, and then behaviors for performing the action and verifying the results.
- Define a *continuously valued parameter adjustment*, as an arbitration between any number of opinions for a continuously valued parameter, followed by a behavior (of any kind) that makes use of that parameter.

This line of reasoning is approximate, but it may support several interesting classes of instruction. For example, the user could construct a *move-in-direction* behavior by instantiating the action part of a *continuously valued parameter adjustment* with *move*, and the arbitration part with estimates of the appropriate heading. Or, more powerfully, he could instruct the system to mate the concept of a *careful action* with the concept of moving through a narrow space. The resulting program would repetitively check for clearance, move, and check for clearance again. If valuable, this plan could be saved and employed as a component elsewhere.

### 5.5.5. The TeamWorks Operating Environment

We built our Phase I demonstration using the TeamWorks operating environment, implemented at ADS for the Army Tank Command. The TeamWorks system provided a simulation base with multiple vehicles, a town environment to operate within (the town of Singling, France), and a control language for directing vehicles, which we have augmented to make it contain our primitive instruction set. The interested reader is referred to [7] for more detail.

## 5.6. Work Performed

### 5.6.1. The Phase I Contract

When ADS and TACOM agreed on the work to be performed it was decided to demonstrate the feasibility of our ideas by building a demonstration of what the Phase II project might eventually produce. From our perspective, the implementation focus gave the project a somewhat "bottom up" flavor. The budget constraints (12 man-weeks) were such that we were forced to work with available tools and techniques. Within this framework, we demonstrated a vertical slice through the capabilities discussed in our proposal, i.e. we addressed a piece of each of the following problems:

- reactive execution -- showing agents acting according to a plan but responding to real world interruptions;
- universal plan control structure -- having a robust set of reactions which allow plan completion despite arbitrary intervention;
- behavioral programming -- the use of programming primitives that coordinate sensing and action to provide "domain relevant" behavior;
- instructability -- the ability to instruct robots/vehicles/agents in behavioral terms, rather than having to teleoperate them;
- interactive sensor interpretation -- use of a person to perform sensor processing tasks that cannot be automated with current technology.

Toward these ends, we took the following sequence of steps.

1. We identified a small number of example behaviors that provide opportunity for telerobotic operators to instruct the vehicles, and that capture responses to contingencies.
2. For lack of a Universal Plans interpreter we implemented the chosen behaviors as a Lisp program behaviorally similar to a Universal Plan.
3. We augmented the TeamWorks environment (as produced by a Phase I SBIR, i.e. a simple simulator for multiple robotic vehicles) so that it provided
    - the perception data required by the behaviors controlling the vehicles;
    - an interface permitting telerobotic instruction to individual vehicles;
    - an interface permitting the user to create contingencies whenever and wherever s/he saw fit.
4. Having gotten some behaviors up and running on the vehicles, we modified the behaviors and the TeamWorks interface to extend the degree of instructability supported.

5. We built a demonstration of the vehicles performing the chosen behaviors, with contingencies and some instructability, within the TeamWorks environment. This demonstration was the focal point of the final review meeting.

6. We also delivered two interim progress reports.

### 5.6.2. The Demonstration Software

ADS demonstrated to TACOM an interface on the MAC II that allowed one person to manage four vehicles concurrently, in a narrow range of tasks that required navigation. This was possible because the vehicles were endowed with a measure of reactive intelligence -- they detected obstacles in their path, and contingencies that arose along their way, and the vehicles not only took steps to avoid collisions but (at the same time) asked the human operator what to do about such things. As a result, it became unnecessary for the operator to watch the vehicles closely.

Below we first describe how navigation worked, then show how it was possible to navigate reactively in the service of a multi-step plan.

To initiate navigation the human operator needed to specify only a point toward which the vehicles should move, and s/he could then sit back and watch how the vehicles behaved. Obstacles included trees and buildings (which could conceivably be marked on a map) and also bomb craters, which could not be mapped in advance. To make that point strongly, the person commanding the vehicles in the software demonstration was allowed to create bomb craters whenever and wherever s/he saw fit, and the vehicles would still navigate around them (see Figure 5). In other words, reactive behavior allowed the vehicles to navigate in unfamiliar terrain. Furthermore, the response to a new bomb crater was instantaneous: reactivity was much faster than route planning.

Until the vehicles actually arrived at their target point, they required no further help except in situations they could not handle by themselves. Such situations included being confronted head-on by a large obstacle with no obvious detour; being unable to reach the designated point, or making inadequate progress; and detecting enemy aircraft overhead. In such situations the operator needed to provide new instructions, but most of the time the operator had nothing to do other than function as a quality controller on the vehicles' behavior. That was of course the point of our project.

When the vehicles needed help, the control interface would indicate this by opening a small pop-up window that indicated the nature of the problem, and listed some options for the operator to choose from. For example, when confronted by an obstacle and seeing no obvious detour, a vehicle would pop up a window that offered the options

- Continue (i.e. proceed through the obstacle, e.g. if it is only a small tree or a shallow bomb crater); or

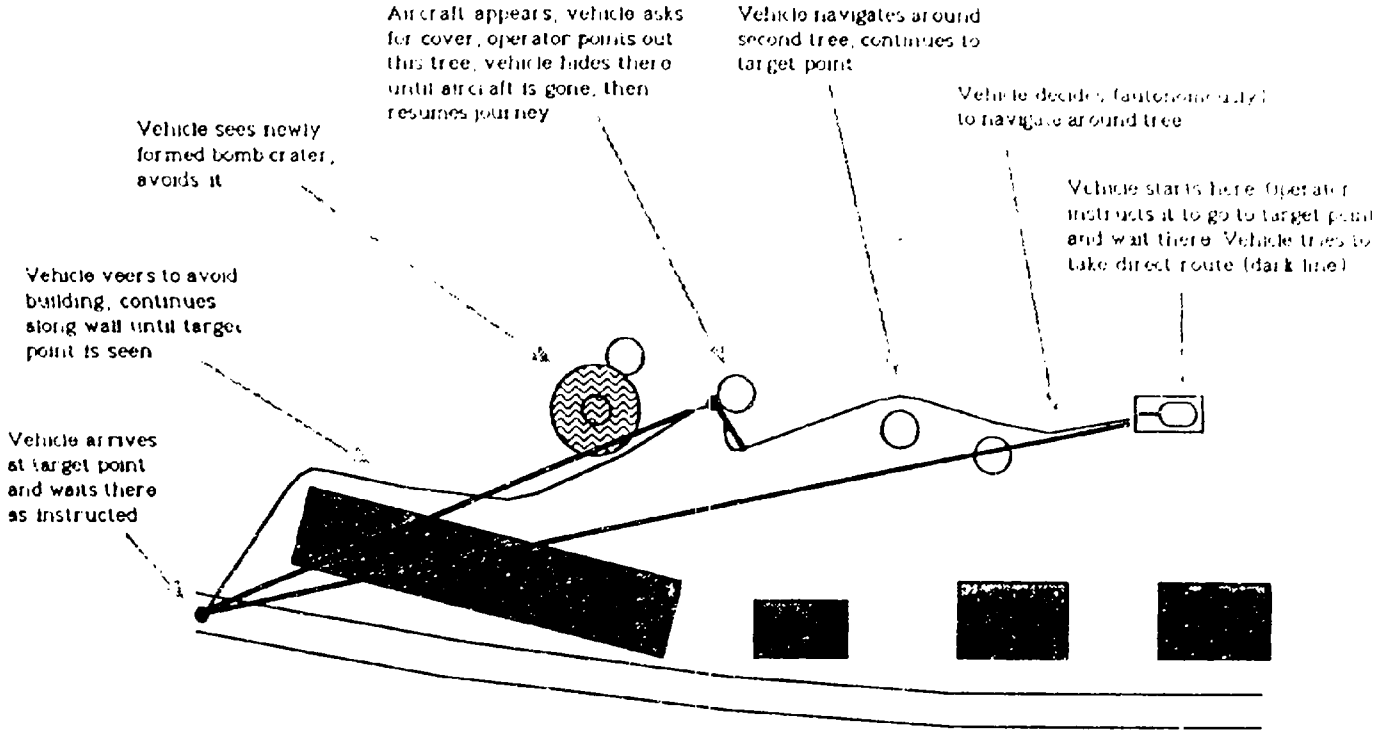


**Figure 5:** Reactively Navigating Around Surprise Obstacles.

- Detour (with sub-options to go right or left around the obstacle); or
- Approach (i.e. move up to the obstacle, with sub-options to end up touching the obstacle, or to ram it hard, or to stand off at some distance, etc).

Even if the operator was too busy to attend to the vehicle's question, or if the operator chose not to select an option, the vehicle did something reasonable, such as stopping just short of the obstacle.

If the operator thought that none of the options offered by the pop-up window were appropriate, s/he could turn off parts of the navigation functions of the vehicles, and could take over control of the vehicles' speed and direction (using the arrow keys on the Apple Macintosh keyboard). In effect, the operator could resort to driving the vehicles (teleoperation) if necessary. This demonstrated that our software would not limit the vehicles' capabilities in unexpected situations.

It also happened that, after a vehicle thought it was in trouble, it occasionally discovered a way to solve its own problem. This could happen in numerous ways, such as having the aircraft overhead going away again, or having the vehicle suddenly discover an alley after it thought it wasn't making any progress. In such cases, the vehicles' reactivity allowed them to retract the question -- the pop-up window would go away by itself -- and the vehicles would resume what they were doing before the problem arose.

For purposes of this demonstration, the communication between the operator and the vehicles was set up so that the operator could only talk to one vehicle at a time. Similarly, only one vehicle could ask for help at any time. This was done to provide for the small screens of most Apple Macintosh IIs, but could have been done differently.

For the demonstration, ADS had prepared a three-step plan of which navigation was only a part. The plan, which we called "wild-ride", required a vehicle to penetrate to the center of an enemy-occupied village, shoot in all directions, and then leave the village. The first and last steps required navigation. The demonstration showed how the vehicle could be side-tracked on the way into the village, without confusing the vehicle about exactly when it should start shooting (only when it had reached the center of the village). That is, the shooting task was initiated only when the vehicle had achieved the context specified for that task. If the vehicle had never made it to the village center it would nevertheless have refused to start shooting prematurely (unless its instructions were changed).

The demonstration program also had some significant weaknesses. Due to the limited time available, the vehicles' intelligence level was only rudimentary. When left to themselves, the vehicles occasionally brushed against the corners of buildings. When the vehicles were in column formation and the lead vehicle backed up, the vehicles behind it

did not know they had to move too. The lead vehicle did not know when it should have waited for the rest of the team. The vehicles did not always take the best route to their destination. In some situations the vehicles could be seen turning from side to side like a pendulum (because the code couldn't make up its mind which way to go). The simulation provided for only one kind of surprise obstacle (bomb crater), only one kind of surprise threat (aircraft overhead), and only one formation (column formation). There was no capability whatsoever for a coordinated team maneuver. And most importantly, the demonstration avoided the problem of having the vehicles do something sensible when they were given conflicting instructions. At one point Mr. Schoppers told the vehicles to follow two paths at the same time. The vehicles followed neither path, crashing into a building instead.

### 5.6.3. Implementation of Demonstrated Behaviors

A behavior is a set of computations that can be activated on an agent. At the current time, each behavior can contain a set of procedures to execute (in parallel), a set of reactions to environmental cues, some termination criteria, a command vocabulary that will only be active when that behavior is active, and a set of behavior transition triggers which invoke other behaviors when various mental or physical conditions become true. A simple example is shown below:

The "Wild-Ride" Behavior

```
(create-task
   '((setq pathplan (list town-center))
     (wait-for '(very-near town-center))
     (have 'new-behavior (startbehavior 'shoot-em-up agent-to-run-on))
     (wait-for '(ask new-behavior done?))
     (setq pathplan (list east-of-town))
     (wait-for '(very-near east-of-town))))
```

This "Wild Ride" behavior consists of a single task which specifies a sequential plan, namely to go to the center of the village, enter shoot-em-up mode, and go to a point east of town once the shoot-em-up behavior terminates.

The shoot-em-up behavior contains two parallel tasks and a reaction:

Contents of the Shoot-em-up behavior

```
(create-task '((perform choose-random-directions)))
(create-task '((perform shoot-at-random-times)))
(create-reaction :test-every-cycle 'threatened?
                 '((yell-for-help)
                   (deactivate)))
```

When activated, the agent will fire his cannon in random directions continuously (choose-random-directions and shoot-at-random-times are tasks which loop forever), while actively testing to see if the "threatened?" condition becomes true. The shooting

will continue indefinitely until either the vehicle is threatened, or its instructions are changed. The reaction to being "threatened?" will take place above and beyond other reactions to "threatened?" that the agent may have acquired from other behaviors.

A behavior can also contain a Universal Plan, which encodes a collection of responses to a situation. Universal Plans are implemented as single step, repeating tasks, in order to satisfy the criterion that all possible reactions be considered at each time cycle (the foundational principle of reactive behavior).

Here follows a translation of the Universal Plan that decides what direction the vehicles should be pursuing at each moment. `target-direction` is the direction from the vehicle to the next location it's been told to reach. `current-direction` is the vehicle's actual direction. `desired-direction` is the output of this Universal Plan, and is set to the direction the vehicle decides to follow (for now). Note that this example merely indicates how a Universal Plan works; the actual Universal Plans language is not shown here. Likewise, the demonstration software did not contain a *bona fide* Universal Plan, but ran a Lisp function that behaved similarly.

```
if null(target-heading)
then desired-heading := current-heading
else if no-obstacle(target-heading)
     then desired-heading := target-heading
     ; there's an obstacle on the way to the target point
     else if close-to(current-heading,target-heading)
          then if got-instruction(obstacle-response)
          .    then if obstacle-response == detour-left
          .    .    then desired-heading := current-heading + 5
          .    .    else if obstacle-response == detour-right
          .    .         then desired-heading := current-heading - 5
          .    .         else desired-heading := current-heading
          .    ; no advice yet about avoiding the obstacle
          .    else if head-on-approach-to-obstacle(target-heading)
          .            then desired-heading := current-heading
          .                 notify-operator(obstacle-dialog)
          .            else desired-heading := obstacle-orientation
          ; now we're already well off the path to the target point,
          ;   so should see about getting back on the track
          else if no-obstacle(current-heading)
                    then if current-heading > target-heading
               .    then if no-obstacle(current-heading - 5)
               .    .    then desired-heading := current-heading - 5
               .    .    else desired-heading := current-heading
               .    else if no-obstacle(current-heading + 5)
               .            then desired-heading := current-heading + 5
               .            else desired-heading := current-heading
               ; we're off track, and there's an obstacle straight ahead too
               else if got-instruction(obstacle-response)
                    then if obstacle-response == detour-left
                    .    then desired-heading := current-heading + 5
                    .    else if obstacle-response == detour-right
                    .         then desired-heading := current-heading - 5
                    .         else desired-heading := current-heading
                    ; no advice yet about avoiding this obstacle
                    else if head-on-approach-to-obstacle(current-heading)
                          then desired-heading := current-heading
                               notify-operator(obstacle-dialog)
                          else desired-heading := obstacle-orientation
```

On each reaction cycle, this "decision tree" is executed to reach a decision about a suitable **desired-heading**. Clearly, there is no relationship between the outcomes of successive decision cycles: No matter what decision was made on one cycle, every possible outcome is again available on the next cycle.

Note that the sequential plan for Wild Ride (shown above) has very little of the reactive flavor which is evident in the Universal Plan above. The agent is restricted to setting a goal, invoking a behavior, and waiting for a response, or in the case of the shoot-em-up behavior, explicitly installing a single reaction. It would be possible in principle to express Wild Ride as a Universal Plan, meaning a set of reactions which would diagnose

the current situation at each time step (e.g., find the location of the agent in the plan) and select the appropriate response (often, to do nothing). However, it is not clear that such flexibility is desired given that Wild Ride is, in concept, a sequential task that is very unlikely to unfold in a different order from the one specified. Nevertheless, certain things might go wrong: the agent might exceed time or fuel constraints, or it might be unable to reach the destination, etc. In his recent thesis, Firby [2] has examined types of reactivity that are desireable in sequential plans. We hope to build *sequential behavior* types which incorporate aspects of this work in the future.

## 5.7. Future Work

The work done in Phase I was a quick, vertical cut on capabilities that need to be vastly extended to build a useful system. In particular, the demonstration software had some significant weaknesses. Those weaknesses fall into categories:

- problems that are solvable with current tools and technology, and that require only more time.
- problems that arose because our tools were inadequate, and that can be solved only by rebuilding those tools.
- problems that require new technology.

Problems solvable with current tools and technology included the following. These problems will be dealt with on the larger Phase II effort.

1. There was no communication between vehicles, and only limited concern for where the other vehicles were. Thus, when the vehicles were in column formation and the lead vehicle backed up, the vehicles behind it were not programmed to realize that they had to move too. Similarly, the lead vehicle did not care whether it should have waited for the rest of the team.
2. The vehicles did not always take the best route to their destination. Sometimes they moved straight past a turn that would clearly have gotten them to their destination earlier. In general, the vehicles got by with a minimum of sensory data (usually range measurements).
3. The vehicles occasionally brushed against the corners of buildings. This happened because the navigation algorithm that controlled the vehicles was quite rudimentary.
4. In some situations the vehicles could be seen turning from side to side like a pendulum. The code was not engineered for the situation that led to this behavior: the vehicle was heading into a wedge-shaped alley between two buildings, and was trying to turn away from both walls, hence swung to the left to avoid the wall on the right, then saw the wall on the left and swung back to the right.

5. The simulation provided for only one kind of surprise obstacle (bomb craters), only one kind of surprise threat (aircraft overhead), and only one formation (column formation). Alternatives could be provided for each of these, along with suitable responses.

Problems requiring the rebuilding of our tools center around the inadequacy of TeamWorks I as a testbed environment. These problems will be rectified by the TeamWorks II effort, not in this Phase II effort.

1. The simulated vehicles were devoid of dynamic realism -- they could change their speed and direction by arbitrarily large amounts, instantaneously.
2. The only things to control were vehicle speed and direction. Vehicles did not have engines, gear-trains, brakes, etc.
3. There was no protocol for communicating with the vehicles, and no bandwidth limitation between the vehicles and our software. Anything at all could be communicated, instantaneously.
4. The only way to detect obstacles in the simulated environment was by sampling the color of the picture pixels. However, the presence of other vehicles could *not* be detected in the same way.
5. The simulated environment was flat, two-dimensional. There was no place for exploiting the defensive utility of hills, for example.
6. The simulation provided only four tanks, and increasing that number was exceedingly difficult (e.g. to simulate combat between two tank platoons, or even, to introduce a single enemy tank).
7. The simulation was not real-time. That is, even if the vehicles did arbitrarily large amounts of reasoning, the world still only changed by one time step.

Problems requiring new technology were entirely avoided during the Phase I work, but will be central in the Phase II effort:

1. The Phase I software had no capability whatsoever for a coordinated team maneuver. Not only was it impossible to communicate between vehicles, but the language available for instructing the vehicles was extremely limited. Solving this problem requires an implementation of Shapiro's Reactive Architecture (see Section 5.5.2).
2. For the vehicles to achieve significant competence, we will not only have to refine the few behaviors we have already implemented, but will have to develop a large number of new behaviors, and will also have to endow the vehicles with a capability for choosing intelligently from several alternative behaviors.
3. We envisage a control interface in which a vehicle's activities and "mental

state" are laid open to the operator, who can then turn behaviors on and off, start new behaviors in parallel or as replacements for others, and even change the vehicle's entire plan, at any moment. We have demonstrated this in our ability to control some or all of a vehicle's speed, direction, communication, and progress monitoring, turning these activities on and off as we saw fit. For Phase II we have not only to design a more general interface, but also to discover principles of concurrent activity that will streamline the operator's instruction task.

4. Phase I avoided the problem of having the vehicles do something sensible when they were given conflicting instructions. The problem is trivial if one knows in advance what behaviors will be performed simultaneously -- one simply engineers the interactions away -- but this is an entirely new area when one knows nothing about what might happen concurrently. Not only must the interactions be detected, but then one must decide how to respond. The detection can be done either at vehicle instruction time, or at behavior execution time. Both are open problems. We expect the solution to involve the automatic analysis of multiple Universal Plans so as to create a single executable Behavior. The symbolic nature of Universal Plans will be a key ingredient in this solution.

To summarize, where in Phase I we implemented a minimal set of intelligent functions and prototyped part of a suitable control interface, in Phase II we intend to:

- enlarge the competence of the robotic vehicles by building up the library of behavior options, and by equipping them to make sensible choices between the options they know about,
- extend the behavior representation languages and component technologies so that more complex behaviors can be assembled by vehicle operators, with automated help, for both individual vehicles and teams of vehicles;
- develop the vehicle control interface to provide a realistic display of the environment and status of each vehicle, and so demonstrate that telerobotic instruction and teleoperation can be carried on concurrently and in near-real time;
- experiment with achieving greater than 3-on-4 manpower leverage, for example by equipping the Wiesel vehicles with inexpensive additional hardware that would significantly ease the burden of the remote driver.

# APPENDIX

## The Review Meeting -- Minutes

Mr. Schoppers, ADS Principal Investigator for this project, presented this work and its Phase II goals to Mr. Sieh (COTR), Mr. Schehr, and approximately six others at TACOM on February 27th 1989. The following report of the meeting serves as the meeting's Minutes, and is included here as required by the Phase I contract.

Before the meeting began, Mr. Schoppers prepared the software for demonstration. The Apple Macintosh II on which the software was to be run had a smaller memory than the one on which the software had been created; to deal with this, Mr. Schoppers turned off the MultiFinder application.

The meeting began at approximately 10:30am. The agenda of the meeting was as follows:

1. Presentation of Phase I effort:
    a. Terms of the Phase I contract.
    b. Review of the technologies and tools used.
    c. Achievements and limitations of the work performed.
2. Demonstration of the software developed under Phase I.
3. Discussion of Phase II goals and constraints.

    Opportunity for TACOM personnel to experiment with the demonstration oftware.

Mr. Schoppers moved quickly through the details of the contract and the relevant technologies, focusing on how those technologies applied to TACOM's RCC work. Even so there was some discussion about the ways in which ADS's approach differed from that of the ALV, for example. Mr. Schoppers pointed out three significant differences:

1. Where the ALV was autonomous, ADS is proposing an interactive approach that allows for a spectrum of autonomy from teleoperation to telerobotics.
2. Where the ALV was a lone vehicle, ADS will provide for control of a team of vehicles simultaneously.
3. Where the ALV was an unachievable goal given current technology, ADS, through telerobotic control, will demonstrate manpower leverage in two years.

Having explained what to look for in the demonstration, Mr. Schoppers ran the software as described in Section 5.6.2.

After the demonstration Mr. Schehr, Mr. Sieh and Mr. Schoppers discussed Phase II work. It emerged that, due to a tightening budget and LABCOM's influence on Phase II awards, ADS's proposed Phase II work should be clearly relevant to the Robotic Command Center in the near term. This meant that where previously TACOM and ADS had agreed to work only with technology that was feasible within a ten-year time horizon, we should now shorten our horizon. This would streamline TACOM's approach to LABCOM.

Since the present RCC design has one commander and two drivers controlling four vehicles, with no computer power on the actual vehicles, ADS's demonstration was ahead of its time. While TACOM found our proposed one-person-four-vehicle approach exciting, that force multiplication factor might be considered implausible and/or irrelevant by LABCOM. A similar obstacle was the fact that the vehicles' only sensors are video cameras for the drivers' benefit, where our approach required at least a directional range finder. I suggested that we solve the problem by proposing that our Phase II software be able to run on the RCC as currently configured, but that we also use a simulation testbed to experiment with larger force multipliers. Thus the output of our Phase II work would be immediately useful, and with some low-cost hardware additions, would indicate directions for RCC evolution. This was deemed acceptable to both TACOM and LABCOM.

Since ADS did not know the status of the TeamWorks II proposal and did not want to make this TelCon II proposal contingent on the fate of TeamWorks II, Mr. Schoppers suggested a simulation testbed that utilized a Silicon Graphics machine as graphical interface for vehicle control, with remote workstations for vehicle intelligence and physics. ADS already has Silicon Graphics software for 3-d terrain with mobile vehicles and ethernet hookup. They agreed that that was probably the most economical approach under the circumstances, but wanted to economize on the other workstations. If we made do with only one remote workstation, and if TACOM loaned us that one, we would save about $60K for labor rather than hardware costs. The intent was clearly that ADS's Phase II work should focus on developing the competence of the vehicle team, by extending the library of behaviors, by making individual behaviors more sohpisticated, and by building new functionality into the control interface.

The meeting ended at approximately 2:30pm.

# Distribution

Commander
Defense Technical Information Center
Bldg 5, Cameron Station
ATTN: DDAC
Alexandria, Virginia 22304-9990

Manager
Defense Logistics Studies Information Exchange
ATTN: AMXMC-D
Fort Lee, Virginia 23801-6044

Commander
U.S. Army Tank Automotive Command
ATTN: AMSTA-DDL (Technical Library)
Warren, Michigan 48397-5000

Commander
U.S. Army Tank Automotive Command
ATTN: AMSTA-CF (Mr. G. Orlicki)
Warren, Michigan 48397-5000

# References


1. FIKES, R.E. AND NILSSON, N.J. "STRIPS: a new approach to the application of theorem proving to problem solving". *Artificial Intelligence 2* (1971), 189ff.

2. FIRBY, R.J. Adaptive execution in complex dynamic worlds. Ph.D. thesis RR#672, Dept of Computer Science, Yale University, 1989.

3. GEORGEFF, M., LANSKY, A. AND SCHOPPERS, M. Reasoning and planning in dynamic domains: an experiment with a mobile robot. Tech Note 380, AI Center. SRI International, 1986.

4. LAWTON, D. Vision System for Human and Robot Teams. Advanced Decision Systems, 1987.

5. SCHOPPERS, M.J. Universal plans for reactive robots in unpredictable environments. Proc 10th IJCAI, 1987, pp. 852ff.

6. SHAPIRO, D.G. Sniffer: a system that understands bugs. AI Memo 638, MIT, 1981.

7. WALSER, R. TeamWorks Phase I Final Report. Report TR3170-01, Planning Division, Advanced Decision Systems, 1988.